Martyna Majewska 193360

**Adaboost selected multi-class variant classifier for face recognition (Famous48 dataset)**

**Concept & Approach**

The project is based on the core principle of AdaBoost, which involves fitting a sequence of weak learners (e.g., base classifiers like decision trees) on repeatedly re-sampled versions of the data. Each sample is assigned a weight that is adjusted after each training step so that misclassified samples receive higher weights. Samples with higher weights have a greater chance of being selected multiple times in the new data set.

Additionally, the project compares the accuracy of the boosted classifiers and provides the best obtained results. The results were obtained using three files.

**Files**

* **grid\_search.py**
* **classifiers\_comparison.py**
* **adaboost.py**

**Libraries & build-in functions**

* **os** – operating system dependent functionality such as reading or writing to the file system
  + os.getcwd() - returns the current working directory as a string.
  + os.path.join() - joins one or more path components intelligently.
* **numpy** – numerical computing in Python, support for arrays, matrices, and many mathematical functions
* **collections** – alternatives to Python’s general-purpose built-in containers like dict, list etc.
  + defaultdict() - A subclass of the built-in dict class, provides a default value for the dictionary when a key does not exist.
* **matplotlib.pyplot** – Plotting results
* **sklearn** – Machine learning library, providing simple and efficient tools for data mining and data analysis (classifiers, metrics, model selection tools)
  + sklearn.tree – DecisionTreeClassifier()

*max\_depth: the maximum depth of the tree, helps prevent overfitting by restricting the complexity of the model.*

*min\_samples\_leaf: specifies the minimum number of samples required to be at a leaf node, can also help prevent overfitting.*

* + sklearn.ensemble – AdaBoostClassifier()

*n\_estimators: defines the number of boosting stages to be run, how many weak learners (trees) should be combined to form the final strong learner.*

*learning\_rate: contribution of each tree by the learning rate, lower values make the model more robust to overfitting but may require more boosting stages.*

*algorithm: SAMME discrete boosting algorithm.*

* + sklearn.ensemble – RandomForestClassifier()

*max\_depth: Controls the maximum depth of the tree.*

*min\_samples\_leaf: The minimum number of samples required to be at a leaf node.*

*min\_samples\_split: The minimum number of samples required to split an internal node.*

* + sklearn.svm – LinearSVC()

*class supports both dense and sparse input and the multiclass support is handled according to a one-vs-the-rest scheme*

*C: regularization parameter of the LinearSVC, where higher values correspond to less regularization.*

* + sklearn.linear\_model – SGDClassifier()

*alpha: Constant that multiplies the regularization term. The higher the value, the stronger the regularization.*

* + sklearn.linear\_model – LogisticRegression()

*C: inverse of regularization strength.*

* + sklearn.model\_selection – train\_test\_split()

*test\_size/train\_size: represent the proportion of the dataset to include in the test/train split.*

*random\_state: controls the shuffling applied to the data before applying the split.*

* + sklearn.model\_selection – ShuffleSplit()

*n\_splits: number of re-shuffling & splitting iterations.*

*test\_size/train\_size: represent the proportion of the dataset to include in the test/train split.*

*random\_state: controls the shuffling applied to the data before applying the split.*

* + sklearn.model\_selection – GridSearchCV()

*estimator: assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a score function, or scoring must be passed.*

*param\_grid: dictionary with parameters names (str) as keys and lists of parameter settings to try as values, enables searching over any sequence of parameter settings.*

*scoring: strategy to evaluate the performance of the cross-validated model on the test set.*

*n\_jobs: number of jobs to run in parallel, -1 means using all processors*

*verbose: controls the verbosity: the higher, the more messages.*

*Implements a “fit” and a “score” method.*

*Evaluating the results:*

* sklearn.model\_selection – cross\_val\_score()
* sklearn.metrics – accuracy\_score()
* sklearn.metrics – classification\_report(), confusion\_matrix(), ConfusionMatrixDisplay()

*Also used:*

* model.fit(X, y): Trains the model using the provided features (X) and labels (y).
* model.predict(X): Predicts the labels for the provided features (X) based on the trained model.

**Functions developed**

1. **load\_data**(source\_directory, no\_faces)

* Loads dataset, extracting features and labels.
* Filters data based on the specified number of faces.

2. **evaluate\_results**(model, X\_test, y\_test, X\_train, y\_train, cv, name)

* Trains the model and evaluates it using cross-validation.
* Prints classification report and confusion matrix.
* Calculates individual accuracies per person.

3. **prepare\_report**(y\_test, y\_pred)

* Prints classification report and plots confusion matrix.

4. **grid\_search\_adaboost**(classifier, classifier\_name, param\_grid, X\_train, y\_train, X\_test, y\_test)

* Performs grid search for optimal parameters of AdaBoost with specified classifier.
* Displays best parameters and scores.
* Prints classification report and confusion matrix.

*grid\_search.py*

**Customizing the model**

File is responsible for customizing hyperparameters of machine learning models to given dataset. Approach is to objectively search different values for model hyperparameters and choose a subset that results in a model that achieves the best performance on a given dataset. This is called **hyperparameter optimization** or hyperparameter tuning and is available in the scikit-learn Python machine learning library.

Due to huge time complexity, **GridSearchCV** function is limited with cross validation and use first 10 faces from given dataset.

**Description of Actions**

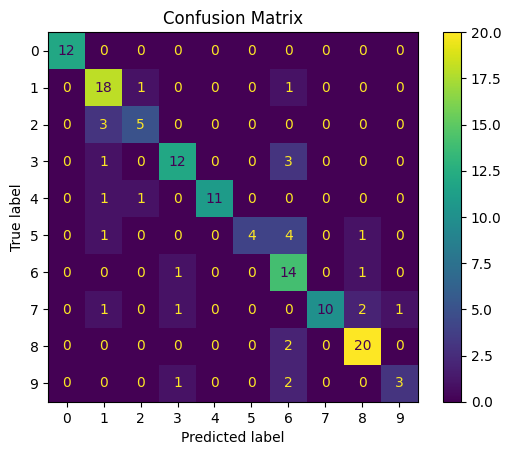
1. Loading dataset using **load\_data1** function from adaboost.py to extract features and labels.

2. Splitting data into training and testing sets - **train\_test\_split** function.

3. Performs grid search for optimal parameters for **Decision Tree, Linear SVC, Logistic Regression, Random Forest, and SGD Classifier** - **grid\_search\_adaboost2** function.

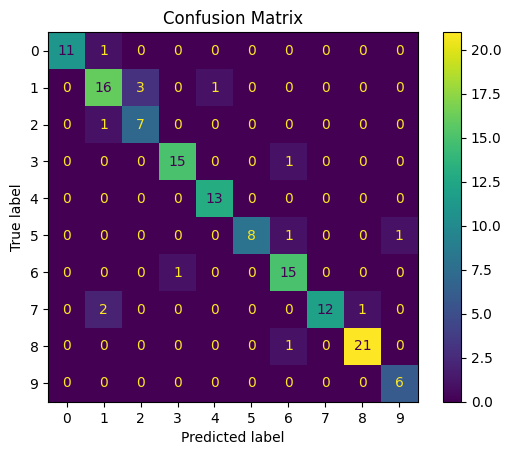
4. Displaying best parameters and scores for each classifier. (**.fit, .predict, prepare\_report3**)

**Fitting parameters for 10 faces**

Mean number of photos of each person: 137.9  
Shape of training data: (1241, 576)  
Shape of training label: (1241,)  
Shape of test data: (138, 576)  
Shape of test label: (138,)

1. **DecisionTreeClassifier**

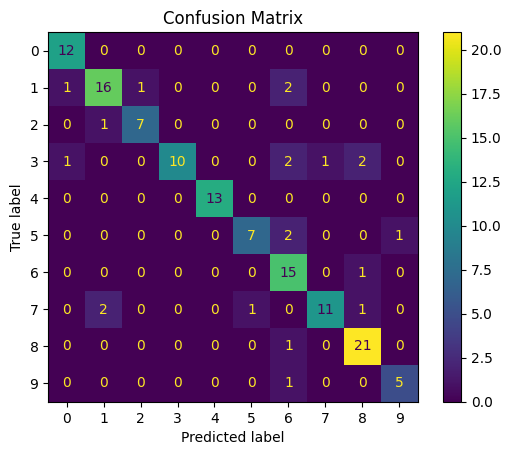
Fitting 5 folds for each of 8 candidates, totalling 40 fits

Best parameters for DecisionTree: {'estimator\_\_max\_depth': 4, 'estimator\_\_min\_samples\_leaf': 5, 'learning\_rate': 0.5, 'n\_estimators': 500}  
Best Score for DecisionTree: 0.828381267003498

1. **LogisticRegression**

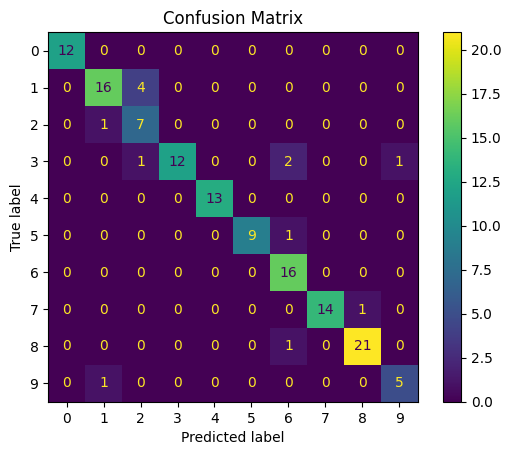
Fitting 5 folds for each of 48 candidates, totalling 240 fits

Best parameters for LogisticRegression: {'estimator\_\_C': 100, 'learning\_rate': 0.01, 'n\_estimators': 50}  
Best Score for LogisticRegression: 0.894448762793108



1. **RandomForestClassifier**

Fitting 5 folds for each of 48 candidates, totalling 240 fits

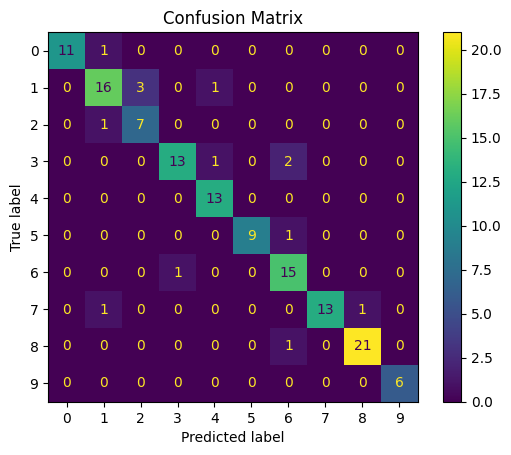
Best parameters for RandomForestClassifier: {'estimator\_\_max\_depth': None, 'estimator\_\_min\_samples\_leaf': 1, 'estimator\_\_min\_samples\_split': 4, 'learning\_rate': 0.1, 'n\_estimators': 500}  
Best Score for RandomForestClassifier: 0.8478260869565217

1. **LinearSVC**

Fitting 5 folds for each of 16 candidates, totalling 80 fits

Best parameters for LinearSVC: {'estimator\_\_C': 100, 'learning\_rate': 0.01, 'n\_estimators': 500}

Best Score for LinearSVC: 0.9057971014492754

****

1. **SGDClassifier**

Fitting 5 folds for each of 168 candidates, totalling 840 fits

Best parameters for SGDClassifier: {'estimator\_\_alpha': 0.001, 'estimator\_\_penalty': 'l2', 'learning\_rate': 0.01, 'n\_estimators': 500}

Best Score for SGDClassifier: 0.8985507246376812

Based on these results for given 10 faces, we will use the LinearSVC classifier for our dataset as it delivered the best performance – about 90% accuracy.

The confusion matrixes shows that across all evaluated classifiers, the best accuracy is consistently obtained for person no. 8, while the worst accuracy is consistently for person no. 9. Among the classifiers, the DecisionTreeClassifier, used as the default estimator for AdaBoost, performed the worst overall.

*classifiers\_comparison.py*

**Description of Actions**

1. Loading dataset using **load\_data1** function from *adaboost.py* to extract features and labels.

2. Splitting data into training and testing sets - **train\_test\_split** function.

3. Classifier Comparison - training various classifiers (AdaBoost with different base classifiers) on the training set. (**.fit**) 4. Evaluates accuracy of different classifiers on the whole famous48 dataset (**.predict, accuracy\_score**). 5. Plotting accuracy of different base classifiers in AdaBoost. (**matplotlib.pyplot**)

**Products**

Mean number of photos of each person: 142.39583333333334

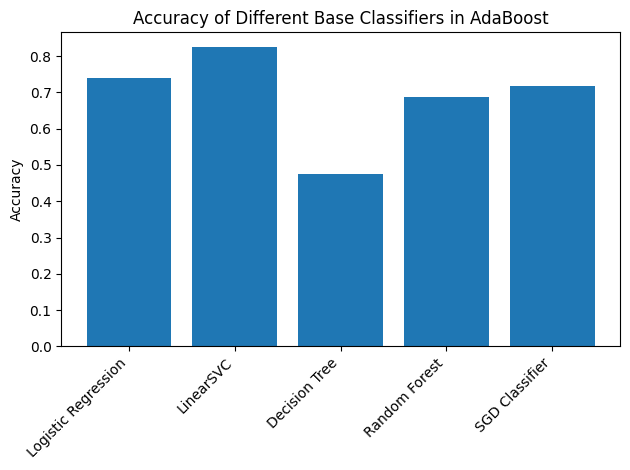
Shape of training data: (6151, 576)

Shape of training label: (6151,)

Shape of test data: (684, 576)

Shape of test label: (684,)

Classifier: Logistic Regression

Model Evaluation - accuracy: 0.7412280701754386

Classifier: LinearSVC

Model Evaluation - accuracy: 0.8245614035087719

Classifier: Decision Tree

Model Evaluation - accuracy: 0.47514619883040937

Classifier: Random Forest

Model Evaluation - accuracy: 0.6871345029239766

Classifier: SGD Classifier

Model Evaluation - accuracy: 0.7178362573099415

For given dataset the best accuracy is obtained with **LinearSVC** classifier with parameters shown below.

AdaBoostClassifier(estimator = LinearSVC(C=100), learning\_rate = 0.01, algorithm='SAMME')

*adaboost.py*

**Description of Actions**

1. Loading dataset using **load\_data1** function from *adaboost.py* to extract features and labels.

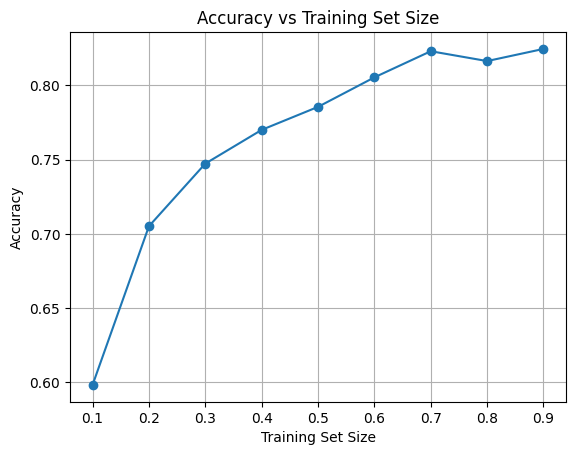
2. Model Training - fitting models to training data (**train\_test\_split**).

3. Evaluation using cross-validation, displaying accuracy per person and overall model accuracy (**evaluate\_results2**, **prepare\_report3**).

4. Varying Training Set Sizes from 10% to 90% and evaluating performance (repeating points 2 & 3).

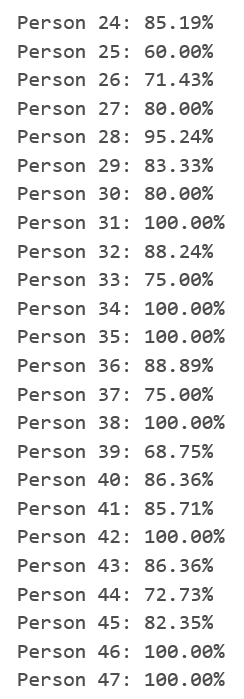
5. Plotting accuracy vs training set size of Adaboost with LinearSVC (**matplotlib.pyplot**).

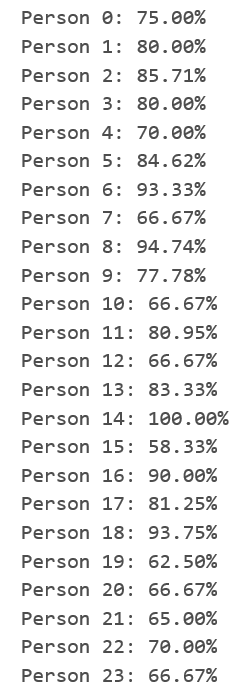
**Products**



**Results: Performance on the 48 faces (dataset)**

Mean number of photos of each person: 683.5  
Shape of training data: (6151, 576)  
Shape of training label: (6151,)  
Shape of test data: (684, 576)  
Shape of test label: (684,)

  
On average, LinearSVC model has f1 score of 0.805533 +/- 0.015 on the training set.  
Accuracy per each person (label) in percentage:



**Model Evaluation - accuracy: 0.8245614035087719**

The summary of accuracies obtained by the hyperparameter-tuned classifiers on the entire dataset reveals that the LinearSVC classifier using 90% of the dataset as a training size performed the best.